Error Handling and Logging

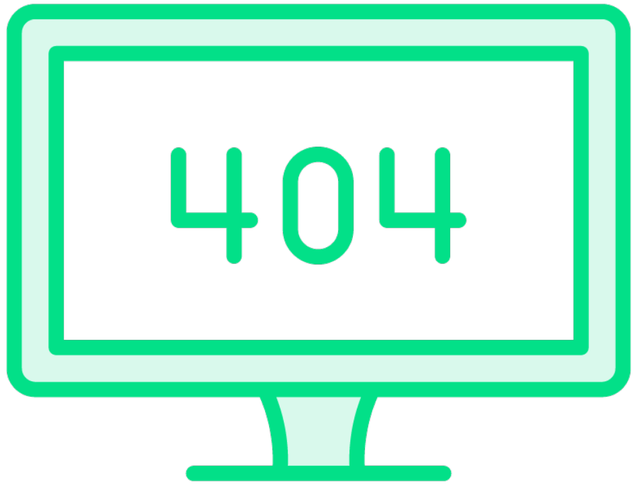
Paul Mooney
Chief Software Architect, Microsoft MVP

@daishisystems | www.insidethecpu.com

Overview

Coming Up

- Enhance security, user experience with error handling
- Implement efficient logging for web monitoring
- Gain insights into code performance, behavior
- Understand logging's role in web security
- Utilize popular logging libraries effectively
- Handle errors using types, techniques
- Leverage error interface, fmt package
- Grasp panic, recover concepts appropriately
- Learn when, how to use



Error Handling

- Built-in error type for abnormal states
- Non-nil error value signifies occurrence
- Graceful recovery without crashes or issues

Error Handling Simple Example

```
func main() {
    if err != nil {
        // handle the error
    }
}
```

Custom Error Types

```
if f < 0 {
    return 0, errors.New("math: square root of negative number")
}
//If an error has occurred print it
if err != nil {
    fmt.Println(err)
}
```

Formatted Error Strings

```
if f < 0 {
    return 0, fmt.Errorf("math: square root of negative number %g", f)
}
```

Demo

Effective Error Handling

- Caution: Avoid exposing sensitive error information
- Prevent leaking debugging details in errors
- Go provides panic, recover, and defer
- Panicked state halts regular execution
- Defer statements execute before exiting
- Recover allows resuming normal execution

Logging

- Manage logging within the application
- Use a master routine for logging
- Avoid logging debugging info and stack traces

Demo

Effective Logging

- Code with login scenario
- Logging for login events
- Step-by-step breakdown

Logging Best Practices

- Log events (success/failure)
- Validate inputs, log failures
- Track authentication attempts
- Log access failures
- Monitor data changes
- Track session token attempts
- Log system exceptions
- Record security changes
- Monitor TLS failures

Advanced Logging

- Built-in log package lacks essential features
- Third-party packages enhance Go's logging
- Logrus: Leveled logging, structured logging
- Glog: Leveled logging, rotation, timestamping
- Loggo: Simple, flexible, leveled logging
- Fatal and Panic functions have distinctions

More Logging Best Practices

- Restrict log access to authorized individuals
- Establish a log analysis mechanism
- Prevent execution of untrusted data
- Go's Garbage Collector handles memory cleanup
- Ensure log validity and integrity
- Use cryptographic hash functions for logs

Demo

Ensuring Log File Integrity

- Code for log file integrity verification
- SHA256 hashing method
- Step-by-step breakdown

Summary

Error Handling and Logging

- Logging enh`ances web security, identifies vulnerabilities
- Logging libraries like logrus, zap simplify logging
- Effective error handling improves application behavior
- Go provides error types and handling techniques
- Built in Error interface
- Panic and recover handle exceptional scenarios